CS 189 Introduction to Machine Learning
Spring 2024 Jonathan Shewchuk

HW1

Due: Wednesday, January 24 at 11:59 pm

This homework comprises a set of coding exercises and a few math problems. While we have you train models across three datasets, the code for this entire assignment can be written in under 250 lines. Start this homework early! You can submit to Kaggle only twice a day.

Deliverables:

1. Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below).

2. Submit a **PDF** of your homework, **with an appendix listing all your code**, to the Gradescope assignment entitled "HW1 Write-Up". You may typeset your homework in LaTeX or Word or submit neatly handwritten and scanned solutions. **Please start each question on a new page.** If there are graphs, include those graphs in the correct sections. **Do not** put them in an appendix. We need each solution to be self-contained on pages of its own.

   - On the first page of your write-up, please list students who helped you or whom you helped on the homework. (Note that sending each other code is not allowed.)
   - On the first page of your write-up, please copy the following statement and sign your signature next to it. (Mac Preview, PDF Expert, and FoxIt PDF Reader, among others, have tools to let you sign a PDF file.) We want to make *extra* clear the consequences of cheating.

     "I certify that all solutions are entirely in my own words and that I have not looked at another student's solutions. I have given credit to all external sources I consulted."

3. Submit all the code needed to reproduce your results to the Gradescope assignment entitled "HW1 Code". You must submit your code twice: once in your PDF write-up (above) so the readers can easily read it, and again in compilable/interpretable form so the readers can easily run it. **Do NOT include any data files we provided.** Please include a short file named README listing your name, student ID, and instructions on how to reproduce your results. Please take care that your code doesn't take up inordinate amounts of time or memory.

   The Kaggle score will not be accepted if the code provided a) does not compile or b) compiles but does not produce the file submitted to Kaggle.

Python Configuration and Data Loading

This section is only setup and requires no submitted solution. Please follow the instructions below to ensure that your Python environment is configured properly, and you are able to successfully load the data provided with this homework. For all coding questions, we recommend using Anaconda for Python 3.

(a) Either install Anaconda for Python 3, or ensure you're using Python 3. To ensure you're running Python 3, open a terminal in your operating system and execute the following command:

```
python --version
```

Do not proceed until you're running Python 3.

(b) Install the following dependencies required for this homework by executing the following command in your operating system's terminal:

```
pip install scikit-learn scipy numpy matplotlib
```

Please use Python 3 with the modules specified above to complete this homework.

(c) You will be running out-of-the-box implementations of Support Vector Machines to classify three datasets. You will find a set of `.npz` files in the `data` folder for this homework. Each `.npz` file will load as a Python dictionary. Each dictionary contains three fields:

- **training_data**, the training set features. Rows are sample points and columns are features.
- **training_labels**, the training set labels. Rows are sample points. There is one column: the labels corresponding to rows of training_data above.
- **test_data**, the test set features. Rows are sample points and columns are features. You will fit a model to predict the labels for this test set, and submit those predictions to Kaggle.

The three datasets for the coding portion of this assignment are described below.

- **toy-data.npz** is a synthetic dataset with two features (2-dimensional) and two classes. The training set has 1,000 examples, and no test set is provided. This dataset is only used in Section 2 of this homework.
- **mnist-data.npz** contains data from the MNIST dataset. There are 60,000 labeled images of handwritten digits for training, and 10,000 for testing. The images are flattened grayscale, 28×28 pixels. There are 10 possible labels for each image, namely, the digits 0–9.

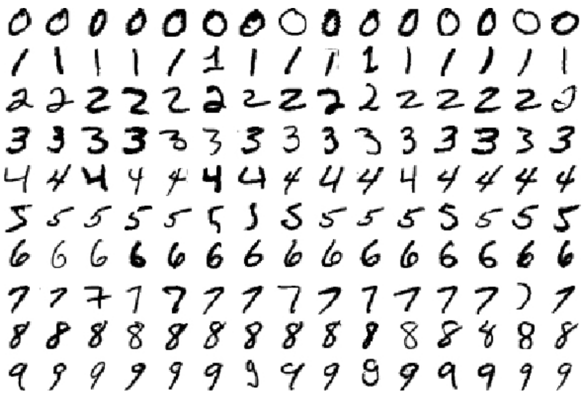


Figure 1: Examples from the MNIST dataset.

- **spam-data.npz** contains featurized spam data. The labels are 1 for spam and 0 for ham. The `data` folder includes the script **featurize.py** and the folders **spam**, **ham** (not spam), and **test** (unlabeled test data); you may modify **featurize.py** to generate new features for the spam data.

To check whether your Python environment is configured properly for this homework, ensure the following Python script executes without error. Pay attention to errors raised when attempting to import any dependencies. Resolve such errors by manually installing the required dependency (e.g. execute `pip install numpy` for import errors relating to the numpy package).

```python
# This file is in scripts/load.py
import sys
if sys.version_info[0] < 3:
  raise Exception("Python 3 not detected.")
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from scipy import io

if __name__ == "__main__":
    for data_name in ["mnist", "spam", "toy"]:
        data = np.load(f"../data/{data_name}-data.npz")
        print("\nloaded %s data!" % data_name)
        fields = "test_data", "training_data", "training_labels"
        for field in fields:
            print(field, data[field].shape)
```

1 Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature : ____

While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

Solution: [**RUBRIC**: (+1 point) if declared and signed.]

This section provides background information on Support Vector Machines (SVMs) used in this homework. You can choose to focus on the coding sections first and revisit this section later, but make sure that this section precedes the coding questions in your write-up.

2 Theory of Hard-Margin Support Vector Machines

A *decision rule* (or *classifier*) is a function $r : \mathbb{R}^d \to \pm 1$ that maps a feature vector (test point) to $+1$ ("in class") or -1 ("not in class"). The decision rule for linear SVMs is of the form

$$r(x) = \begin{cases} +1 & \text{if } w \cdot x + \alpha \geq 0, \\ -1 & \text{otherwise,} \end{cases} \tag{1}$$

where $w \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$ are the parameters of the SVM. The primal hard-margin SVM optimization problem (which chooses the parameters) is

$$\min_{w,\alpha} \ \|w\|^2 \ \text{ subject to } \ y_i(X_i \cdot w + \alpha) \geq 1, \ \ \forall i \in \{1, \ldots, n\}, \tag{2}$$

where $\|w\| = \sqrt{w \cdot w}$.

We can rewrite this optimization problem by using Lagrange multipliers to eliminate the constraints. (If you're curious to know what Lagrange multipliers are, the Wikipedia page is recommended, but you don't need to understand them to do this problem.) We thereby obtain the equivalent optimization problem

$$\max_{\lambda_i \geq 0} \min_{w,\alpha} \ \|w\|^2 - \sum_{i=1}^{n} \lambda_i (y_i (X_i \cdot w + \alpha) - 1). \tag{3}$$

Note: λ_i must be greater than or equal to 0.

(a) Show that equation (3) can be rewritten as the *dual optimization problem*

$$\max_{\lambda_i \geq 0} \sum_{i=1}^{n} \lambda_i - \frac{1}{4} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j X_i \cdot X_j \ \text{ subject to } \ \sum_{i=1}^{n} \lambda_i y_i = 0. \tag{4}$$

Hint: Use calculus to determine and prove what values of w and α optimize equation (3). Explain where the new constraint comes from.

Solution: Taking the gradient with respect to w and α and setting it to zero, we obtain

$$\begin{aligned} w^* &= \frac{1}{2} \sum_{j=1}^{n} \lambda_j y_j X_j, \\ 0 &= \sum_{i=1}^{n} \lambda_i y_i. \end{aligned}$$

Note that this only obtains a critical point of the function. We can conclude that it is indeed a minimum by checking the second derivative of the function, or by arguing that the function is convex. We will do the latter. First, norms are convex (quadratic form with an identity) and thus $||w||^2$ is convex. Next, we realize that all linear functions are convex. Linear functions added to convex functions remain convex and, thus, the function is convex in w. The function is linear in alpha and, thus, it is also convex in α. We thus conclude that the found values are optima.

The latter equation is our new constraint. Substituting this solution for w^* back into equation (3) and noting that $\sum_{i=1}^n \lambda_i y_i \alpha = 0$, we obtain the objective function of equation (4).

(b) Suppose we know the values λ_i^* and α^* that optimize equation (3). Show that the decision rule specified by equation (1) can be written

$$r(x) = \begin{cases} +1 & \text{if } \alpha^* + \frac{1}{2}\sum_{i=1}^n \lambda_i^* y_i X_i \cdot x \geq 0, \\ -1 & \text{otherwise.} \end{cases} \tag{5}$$

Solution: Simply substitute the optimal $w^* = \frac{1}{2}\sum_{i=1}^n \lambda_i y_i X_i$ into equation (1).

(c) Applying Karush–Kuhn–Tucker (KKT) conditions (See Wikipedia for more information), any pair of optimal primal and dual solutions w^*, α^*, λ^* for a linear, hard-margin SVM must satisfy the following condition:

$$\lambda_i^*(y_i(X_i \cdot w^* + \alpha^*) - 1) = 0 \quad \forall i \in \{1, \ldots, n\}$$

This condition is called *complementary slackness*. Explain what this implies for points corresponding to $\lambda_i^* > 0$.

Solution: Notice the optimal primal solution w^*, α^* must fulfill the constraint $y_i(X_i \cdot w + \alpha) \geq 1$. in equation (2). This means if $\lambda_i^* > 0$, then $y_i(X_i \cdot w^* + \alpha^*) - 1$ must equal 0 for complementary slackness to hold.

(d) The training points X_i for which $\lambda_i^* > 0$ are called the *support vectors*. In practice, we frequently encounter training data sets for which the support vectors are a small minority of the training points, especially when the number of training points is much larger than the number of features. Explain why the support vectors are the only training points needed to evaluate the decision rule.

Solution: Every training point X_i that is not a support vector has $\lambda_i^* = 0$, so X_i makes no contribution to equation (5). Only the support vectors contribute to equation (5).

However, every training point that is not a support vector has veto power, in the sense that a hard-margin SVM must classify that training point correctly. (Otherwise, that training point would have been a support vector.)

(e) The obtained parameters when fitting the linear SVM to the 2D synthetic dataset found in **toy-data.npz** approximately correspond to

$$w = \begin{bmatrix} -0.4528 \\ -0.5190 \end{bmatrix} \quad \text{and} \quad \alpha = 0.1471. \tag{6}$$

Using only matplotlib basic plotting functions, in your write-up, produce a plot of

- the data points,
- the decision boundary,
- the margins, defined as $\{x \in \mathbb{R}^2 : w \cdot x + \alpha = \pm 1\}$.

In this plot, where are the support vectors?

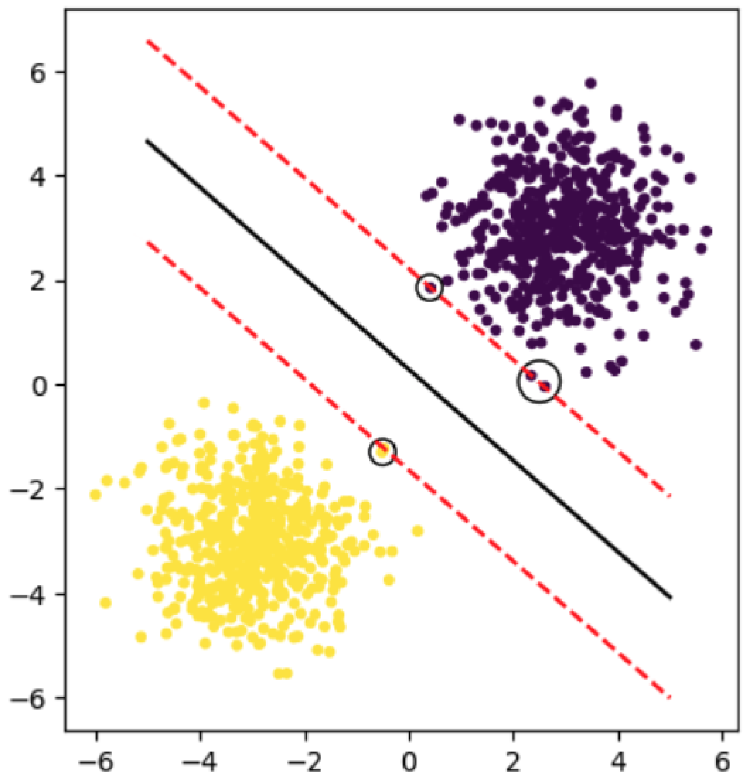
Hint: You can use the following snippet, that plots the data points and decision boundary but not the margins.

```
plt.scatter(data[:, 0], data[:, 1], c=labels)

# Plot the decision boundary
x = np.linspace(-5, 5, 100)
y = -(w[0] * x + b) / w[1]
plt.plot(x, y, 'k')

# Plot the margins
## TODO
```

Solution:



(f) Assume the training points X_i and labels y_i are linearly separable. Using the original SVM formulation (not the dual) prove that there is at least one support vector for each class, +1 and −1.

Hint: Use contradiction. Construct a new weight vector $w' = w/(1+\epsilon/2)$ and corresponding bias α' where $\epsilon > 0$. It is up to you to determine what ϵ should be based on the contradiction. If you provide a symmetric argument, you need only provide a proof for one of the two classes.

Solution: Without loss of generality, consider the $+1$ class. Assume there is no support vector for class $+1$. This means that every training point is beyond the margin, ie $X_i \cdot w + \alpha > 1$ for all i with $y_i = +1$. Let epsilon $\epsilon > 0$ be the smallest distance from the margin for these points, or $\epsilon = \min_i X_i \cdot w + \alpha - 1$.

We can now choose a new weight vector w' and bias α' that achieves lower cost. There are two ways of doing this.

Method 1: Constructing a support vector. Per what we are trying to prove, we know a support vector is likely to be in the better solution. We can make the point closest to the constraint boundary X_j a support vector by assuming $X_j \cdot w' + \alpha' = 1$. As stated in the hint, let $w' = w/(1+\epsilon/2)$. We now need to find the corresponding value of α' by enforcing that condition that $X_j \cdot w' + \alpha' = 1$ and $X_j \cdot w + \alpha = 1 + \epsilon$.

$$\alpha' = 1 - X_j \cdot w' = 1 - \frac{X_j \cdot w}{1+\epsilon/2} = \frac{1+\epsilon/2}{1+\epsilon/2} - \frac{1+\epsilon-\alpha}{1+\epsilon/2} = \frac{\alpha - \epsilon/2}{1+\epsilon/2}$$

Method 2: Enforcing the feasibility constraints. We know that final solution (w', α') must be feasible. Thus, we can determine the required value of α' by enforcing feasibility. First, for the positive points we need $X_i \cdot w' + \alpha' \geq 1$ and we know by assumption that $X_i \cdot w + \alpha \geq 1 + \epsilon$ or $X_i \cdot w \geq 1 + \epsilon - \alpha$. Then:

$$X_i \cdot w' + \alpha' = \frac{X_i \cdot w}{1+\epsilon/2} + \alpha' \geq \frac{1+\epsilon-\alpha}{1+\epsilon/2} + \alpha' \quad \text{should be} \geq 1$$

Second, for the negative points we need $X_i \cdot w' + \alpha' \leq -1$ and we know that $X_i \cdot w + \alpha \leq -1$ or $X_i \cdot w \leq -1 - \alpha$. Then:

$$X_i \cdot w' + \alpha' = \frac{X_i \cdot w}{1+\epsilon/2} + \alpha' \leq \frac{-1-\alpha}{1+\epsilon/2} + \alpha' \quad \text{should be} \leq -1$$

Rearranging the two desired inequalities, gives the following bounds on α':

$$\alpha' \geq 1 - \frac{1+\epsilon-\alpha}{1+\epsilon/2} = \frac{\alpha-\epsilon/2}{1+\epsilon/2}$$
$$\alpha' \leq -1 - \frac{-1-\alpha}{1+\epsilon/2} = \frac{\alpha-\epsilon/2}{1+\epsilon/2}$$

Luckily, these bound are tight! So we know that $\alpha' = \frac{\alpha-\epsilon/2}{1+\epsilon/2}$ and that this is the only value that would work.

Now we must show that given there is no support vector, we have derived a better solution (w', α'). First, this solution has a lower cost as $\|w'\| = \|w\|/(1+\epsilon/2) < \|w\|$. Next we must show that it is feasible.

For all points where $y_i = +1$, we have the following:

$$X_i \cdot w' + \alpha' = \frac{X_i \cdot w + \alpha - \epsilon/2}{1+\epsilon/2} \geq \frac{1+\epsilon-\epsilon/2}{1+\epsilon/2} = 1$$

Note that at X_j this inequality is tight by construction. For points where $y_i = -1$, we have:

$$X_i \cdot w' + \alpha' = \frac{X_i \cdot w + \alpha - \epsilon/2}{1 + \epsilon/2} \leq \frac{-1 - \epsilon/2}{1 + \epsilon/2} = -1$$

It thus follows that the new solution is feasible. A symmetric argument shows that this holds for points where $y_i = -1$. Thus, if we do not have a point from each class on either side of the margin, we can reduce the weight vector and get a lower cost solution. It follows that the optimal solution must have at least two support vectors, one for each class.

For the entire assignment, you may use sklearn only for the SVM model. Everything else must be done without the use of sklearn.

3 Data Partitioning and Evaluation Metrics

In machine learning, it is typical to rely on a set of held-out data points, or "validation" dataset, to evaluate the performance of a model and ultimately select the best performing one, while using the rest of the data, or "training" dataset, to train models. In its simplest form, evaluating a trained model requires you to (i) have set aside a validation dataset, and (ii) selected a reasonable metric to evaluate model performance.

In this question, you will implement these components that will be useful for the rest of the assignment. **Please do not use any sklearn functions in this section**.

(a) **Data partitioning**: Rarely will you receive "training" data and "validation" data; usually you will have to partition available labeled data yourself. In this question, you will *shuffle and partition* each of the datasets in the assignment[1]. Shuffling prior to splitting crucially ensures that all classes are represented in your partitions. For this question, please do not use any functions available in sklearn. For the MNIST dataset, write code that sets aside 10,000 training images as a validation set. For the spam dataset, write code that sets aside 20% of the training data as a validation set.

(b) **Evaluation metric**: There are several ways to evaluate models. We will use *classification accuracy*, or the percent of examples classified correctly, as a measure of the classifier performance. Error rate, or one minus the accuracy, is another common metric. Write a function, taking as inputs the set of true labels y and the set of predicted labels $\hat{y}$, that computes the (unweighted) accuracy score s,

$$s = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\left[y_i = \hat{y}_i\right]. \tag{7}$$

Here, $\mathbb{I}\left[y_i = \hat{y}_i\right]$ is an indicator function defined as

$$\mathbb{I}\left[y_i = \hat{y}_i\right] = \begin{cases} 1 & \text{if } y_i = \hat{y}_i \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

n is the total number of input observations, and for any $i \leq n$, y_i and $\hat{y}_i$ respectively denote the ground-truth and predicted label for observation i.

Deliverable: Attach a copy of your data partitioning and evaluation metric code to your homework report under question 3.

Solution: A function that partitions a dataset:

[1]Make sure that you shuffle the labels with the training images. It's a very common error to mislabel the training images by forgetting to permute the labels with the images!

```
def split(data, labels, val_size):
    num_items = len(data)
    assert num_items == len(labels)
    assert val_size >= 0
    if val_size < 1.0:
        val_size = int(num_items * val_size)
    train_size = num_items - val_size
    idx = np.random.permutation(num_items)
    data_train = data[idx][:train_size]
    label_train = labels[idx][:train_size]
    data_val = data[idx][train_size:]
    label_val = labels[idx][train_size:]
    return data_train, data_val, label_train, label_val
```

A function that that implements accuracy:

```
def accuracy(predicted, labels):
    n = len(labels)
    assert n == len(predicted)
    return sum(np.equal(predicted, labels).astype(np.float32)) / n
```

4 Support Vector Machines: Coding

We will use linear Support Vector Machines (SVM) to classify our datasets. For images, we will use the simplest of features for classification: raw pixel brightness values. In other words, our feature vector for an image will be a row vector with all the pixel values concatenated in a row major (or column major) order.

Train a linear SVM on the spam and MNIST datasets. For each dataset, plot the accuracy on the training and validation sets versus the number of training examples that you used to train your classifier. The number of training examples to use are listed for each dataset in the following parts.

You may use `sklearn` only for the SVM model. Everything else must be done without the use of `sklearn`.

(a) For the **MNIST** dataset, use raw pixels as features. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, 5,000, 10,000. For the largest training set, you should expect validation accuracies between 70% and 90%. When you calculate the training accuracy, you only need to calculate on the subset of the data used to train the model, not necessarily the full training dataset.[2]

(b) For the **spam** dataset, use the provided word frequencies as features. In other words, each document is represented by a vector, where the i^{th} entry denotes the number of times word i (as specified in `featurize.py`) is found in that document. Train your model with the following numbers of training examples: 100, 200, 500, 1,000, 2,000, **ALL**. When you calculate the training accuracy, you only need to calculate on the subset of the data used to train the model, not necessarily the full training dataset.

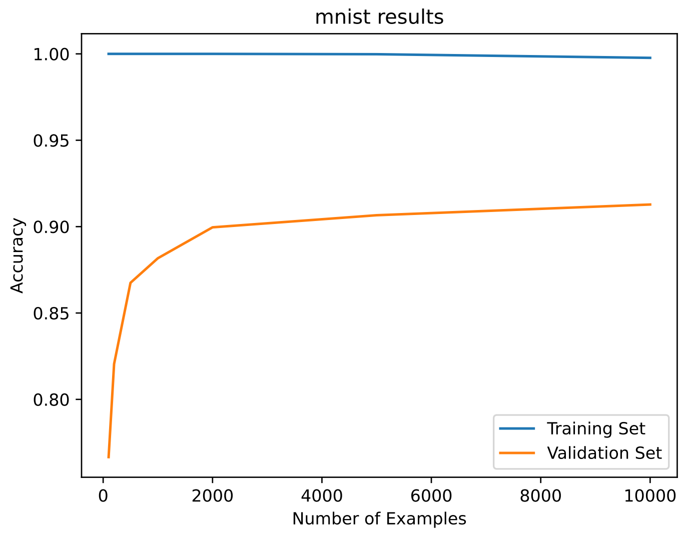
    For the largest training set, you should expect validation accuracies between 70% and 90%.

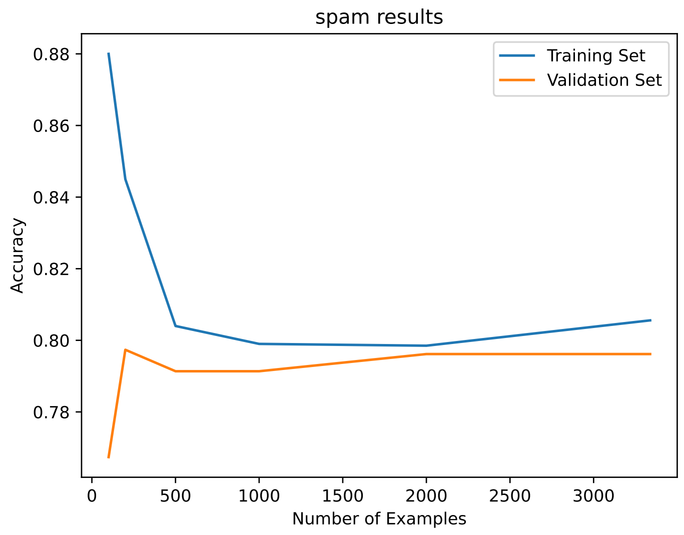
Note: You can use either `SVC(kernel='linear')` or `LinearSVC` as your SVM model, though they each solve slightly different optimization problems using different libraries. On MNIST, `LinearSVC` was faster on one member of Course Staff's laptop, though the exact results will likely depend on your computer, the parameters of the algorithm, and the data (number of data points vs number of features).

Deliverable: For this question, you should include two plots showing number of examples versus training and validation accuracy for each of the datasets. Additionally, be sure to include your code in the "Code Appendix" portion of your write-up.

Solution: Example plots for the MNIST and spam datasets:

[2]Hint: Be consistent with any preprocessing you do. Use either integer values between 0 and 255 or floating-point values between 0 and 1. Training on floats and then testing with integers is bound to cause trouble.





The plot code should look something like this (using matplotlib):

```
def plot_result(self, train_sizes, train_accuracies, val_accuracies, title, filename):
    plt.figure()
    plt.title(title)
    plt.xlabel("Number of Examples")
    plt.ylabel("Accuracy")
    plt.plot(train_sizes, train_accuracies, label="Training Set")
    plt.plot(train_sizes, val_accuracies, label="Validation Set")
    plt.legend()
    plt.savefig(os.path.join(os.path.split(__file__)[0], filename))
```

5 Hyperparameter Tuning

In the previous problem, you learned parameters for a model that classifies the data. Many classifiers also have *hyperparameters* that you can tune to influence the parameters. In this problem, we'll determine good values for the regularization parameter C in the soft-margin SVM algorithm. The interpretation of this parameter, as well as the functioning of the soft-margin SVM will be covered in lecture. For now, consider C as a parameter of a black-box algorithm that we aim to optimize.

When we are trying to choose a hyperparameter value, we train the model repeatedly with different hyperparameters. We select the hyperparameter that gives the model with the highest accuracy on the validation dataset. Before generating predictions for the test set, the model should be retrained using all the labeled data (including the validation data) and the previously-determined hyperparameter.

The use of automatic hyperparameter optimization libraries is prohibited for this part of the homework.

Deliverable: For the MNIST dataset, find the best C value. In your report, list at least 8 C values you tried, the corresponding accuracies, and the best C value. You should try a geometric sequence of C values (not an arithmetic sequence). As in the previous problem, for performance reasons, you are required to train with at least 10,000 training examples. You can train on more if you like, but it is not required. Again, reference any code you used to perform a hyperparameter sweep in the code appendix.

Solution: A function that implements grid search hyperparameter optimization over C.

```
def train(X_train, Y_train, C=1.0):
    model = svm.SVC(kernel='linear', C=C)
    model.fit(X_train, Y_train)
    return model

def hyperopt_experiment(X_train, X_val, Y_train, Y_val, train_size, C_range):
    results = []
    for C in C_range:
        model = train(X_train[:train_size], Y_train[:train_size], C)
        Y_prediction = model.predict(X_val)
        val_accuracy = metrics.accuracy_score(Y_val, Y_prediction)
        results.append((C, val_accuracy))
        print("C", C, "accuracy", val_accuracy)
    return sorted(results, key=lambda x: x[1])[-1][0]
```

6 K-Fold Cross-Validation

For smaller datasets (e.g., the spam dataset), the validation set contains fewer examples, and our estimate of our accuracy might not be accurate—the estimate has high variance. A way to combat this is to use *k-fold cross-validation*.

In k-fold cross-validation, the training data is shuffled and partitioned into k disjoint sets. Then the model is trained on $k-1$ sets and validated on the k^{th} set. This process is repeated k times with each set chosen as the validation set once. The cross-validation accuracy we report is the accuracy averaged over the k iterations.

Use of automatic cross-validation libraries is prohibited for this part of the homework.

Deliverable: For the spam dataset, use 5-fold cross-validation to find and report the best C value. In your report, list at least 8 C values you tried, the corresponding accuracies, and the best C value. Again, please include your code for cross validation or include a reference to its location in your code appendix.

Hint: Effective cross-validation requires choosing from **random** partitions. This is best implemented by randomly shuffling your training examples and labels, then partitioning them by their indices.

Solution: An implementation of cross-validation:

```
def train(X_train, Y_train, C=1.0):
    model = svm.SVC(kernel='linear', C=C)
    model.fit(X_train, Y_train)
    return model

def cv_experiment(training_data, training_labels, k, C_range):
    num_examples = len(training_data)
    idx = self.random.permutation(num_examples)
    x_parts = [None]*k
    y_parts = [None]*k
    part_size = num_examples // k
    for i in range(k):
        si = i * part_size
        if i == k-1:
            ei = num_examples
        else:
            ei = (i+1) * part_size
        x_parts[i] = training_data[idx][si:ei]
        y_parts[i] = training_labels[idx][si:ei]
    assert np.sum(list(map(lambda x: x.shape[0], x_parts))) == num_examples
    assert np.sum(list(map(lambda x: x.shape[0], y_parts))) == num_examples
    results = []
    for C in C_range:
        val_scores = []
        for i in range(0, k):
            X_val, Y_val = x_parts[i], y_parts[i]
            X_train = np.concatenate(x_parts[:i] + x_parts[i+1:], axis=0)
            assert X_train.shape[0] + X_val.shape[0] == num_examples
            Y_train = np.concatenate(y_parts[:i] + y_parts[i+1:], axis=0)
            assert Y_train.shape[0] + Y_val.shape[0] == num_examples
            model = train(X_train, Y_train, C)
            val_scores.append(metrics.accuracy_score(Y_val, model.predict(X_val)))
        results.append((C, np.mean(val_scores)))
        print("C", results[-1][0], "mean val accuracy: ", results[-1][1])
    return sorted(results, key=lambda x: x[1])[-1][0]
```

7 Kaggle

- MNIST Competition: https://www.kaggle.com/competitions/cs189-hw1-mnist-spring-2024
- SPAM Competition: https://www.kaggle.com/competitions/cs189-hw1-spam-spring-2024

With the best model you trained for each dataset, generate predictions for the test sets we provide and save those predictions to .csv files. The csv file should have two columns, `Id` and `Category`, with a comma as a delimiter between them. The Id column should start at the integer 1 and end at the number of elements in the test set. The category label should be a dataset-dependent integer (for MNIST, one of $\{0, \ldots, 9\}$, and for spam, one of $\{0, 1\}$. **Be sure to use integer labels (not floating-point!) and no spaces (not even after the commas).** Upload your predictions to the Kaggle leaderboards (submission instructions are provided within each Kaggle competition). **In your write-up, include your Kaggle name as it displays on the leaderboard and your Kaggle score for each of the three datasets** .

General comments about the Kaggle sections of homeworks. Most or all of the coding homeworks will include a Kaggle section. Whereas other parts of a coding assignment might impose strict limits about what methods you're permitted to use, the Kaggle portions permit you to apply your creativity and find clever ways to improve your leaderboard performance. The main restriction is that you cannot use an entirely different learning technique. For example, this is an SVM homework, so you must use an SVM; you are not permitted to use a neural network or a decision tree instead of (or in addition to) a support vector machine. (You are also not allowed to search for the labeled test data and submit that to Kaggle; that's outright cheating.)

For example, to achieve higher positions on the Kaggle leaderboards, you may optionally add more features or use a nonlinear SVM kernel. Spam is a particularly good dataset for playing with feature engineering; one easy way to perform better in spam/ham is to add extra features with `featurize.py` (see below). Other examples of things you might investigate include SIFT and HOG features for images, and a bag-of-words model for spam/ham. For reasons we'll learn later this semester, dropping features that have little or no predictive power will often improve your test performance as much as adding the right new features. Although extensive creativity isn't generally necessary to get full points on an assignment, topping the Kaggle leaderboard gives your professor good material for letters of recommendation.

Whatever creative ideas you apply, please explain what you did in your write-up. Cite any external sources where you got ideas. If you have any questions about whether something is allowed or not, ask on Ed Discussion.

Remember to start early! Kaggle only permits two submissions per leaderboard per day. To help you format the submission so that Kaggle can interpret it correctly please use `scripts/check.py` to run a basic sanity check.

To check your submission csv,

```
python check.py <competition name, eg. mnist> <submission csv file>
```

Deliverable: Your deliverable for this question has three parts. First, submit to all the Kaggle competitions listed above, and include your Kaggle score in your write-up. Second, include an explanation of what you tried, what worked, and what didn't to improve your accuracy. Finally, make sure to include all the code you used in the code appendix and provide a reference to it.

Modifying features for spam: The Python script `scripts/featurize.py` extracts features from the original emails in the Spam dataset. The spam emails can be found in `data/spam/`, the ham (ie. not spam) emails can be found in `data/ham/`, and the emails for the test set can be found in `data/test/`. You are encouraged to look at the emails and try to think of features you think would be useful in classifying an email as spam or ham.

To add a new feature, modify `featurize.py`. You are free to change the structure of the code provided, but if you are following the given structure, you need to do two things:

- Define a function, eg. `my_feature(text, freq)` that computes the value of your feature for a given email. The argument `text` contains the raw text of the email; `freq` is a dictionary containing the counts of each word in the email (or 0 if the word is not present). The value you return should be an integer or a float.
- Modify `generate_feature_vector` to append your feature to the feature vector. For example:

```
feature.append(my_feature(text, freq))
```

Once you are done modifying `scripts/featurize.py`, re-generate the training and test data by entering the `scripts` directory and running

```
python featurize.py
```

Solution: A function that outputs data formatted for Kaggle submission:

```
def train(X_train, Y_train, C=1.0):
    model = svm.SVC(kernel='linear', C=C)
    model.fit(X_train, Y_train)
    return model

def predict_kaggle(name, training_data, training_labels, test_data, C=1.0):
    model = train(training_data, training_labels, C=C)
    test_labels = model.predict(test_data)

    filename = os.path.join(os.path.split(__file__)[0], '%s_solution.csv' % name)
    f = open(filename, 'w')
    f.write("Id,Category\n")
    for i, y in enumerate(test_labels):
        f.write(str(i + 1) + ',' + str(y) + '\n')
    f.close()
```